ESTRUCTURAS DE CONTROL EN LENGUAJE C

Escuela Académico Profesional de Ingeniería Hidráulica, Universidad Nacional de Cajamarca, Perú, 2026

Milagros Cueva Gálvez ***Llanos Gallardo, Antony Franz*** ***Villena Sanchez, Edward Daniel*** ***evillenas23_1@unc.edu.pe***

Flores Leiva, Yojan Kevin ***Pachamango Miranda, Brayan***

Resumen---Este trabajo presenta el desarrollo y análisis de un proyecto de programación basado en el lenguaje C. Se aplican conceptos fundamentales como variables, estructuras de control, bucles y funciones para la resolución de problemas prácticos. El proyecto tiene como objetivo fortalecer el pensamiento lógico y el razonamiento algorítmico mediante programación estructurada. Los resultados muestran que el uso del lenguaje C permite un control eficiente de los recursos del sistema y contribuye a la comprensión de los fundamentos de la programación de bajo nivel, esenciales en aplicaciones de ingeniería.

Palabras Clave: **lenguaje C, programación estructurada, estructuras de control, funciones, ingeniería**

I. Introducción:

En el presente trabajo práctico, nos enfocaremos en estudiar, practicar y comprender a fondo el lenguaje de programación C, un lenguaje fundamental en el desarrollo de software y sistemas. El objetivo principal es lograr un dominio práctico de su sintaxis y lógica, para lo cual hemos recurrido a una investigación exhaustiva indagando en diversas fuentes de información técnica y documentación oficial.

Nuestra base de estudio se centra en el análisis detallado de los pilares que rigen la lógica de programación en este lenguaje, estructurando el aprendizaje en cuatro áreas críticas:

- Estructuras Secuenciales: Como base del flujo lógico donde las instrucciones se ejecutan de manera lineal.
- Estructuras Condicionales: Para entender cómo el programa toma decisiones basadas en criterios específicos.
- Estructuras Repetitivas: Para dominar la ejecución eficiente de tareas cíclicas o bucles.
- Definición de Funciones: Para aprender a modularizar el código y crear programas más organizados y reutilizables.

A través de este proceso de investigación y práctica, buscamos no solo conocer la teoría, sino también aplicar estos conceptos en ejercicios reales, comparando su funcionamiento con otros lenguajes como Python y evaluando sus capacidades en el entorno tecnológico actual.

II. OBJETIVOS:

a. Objetivo General:

Analizar, implementar y comparar las estructuras fundamentales de programación (secuenciales, condicionales, repetitivas y de funciones) en el lenguaje **C**, con el fin de evaluar su sintaxis, eficiencia, control de bajo nivel y aplicabilidad en el desarrollo de software, contrastándolo con un lenguaje de alto nivel como Python.

b. Objetivo Secundario:

- Explicar conceptualmente y demostrar mediante ejemplos funcionales la sintaxis y el comportamiento de cada tipo de estructura de control en C, haciendo énfasis en el manejo explícito de memoria y tipos de datos.
- Evaluar las ventajas y desventajas inherentes al uso de C para cada estructura, considerando factores como rendimiento, control del hardware, verbosidad del código, y riesgos asociados al manejo directo de memoria.
- Sintetizar las dificultades, aprendizajes y ventajas encontradas durante el proceso, formulando una opinión fundamentada sobre la idoneidad de C como lenguaje base para comprender los fundamentos de la programación y el funcionamiento del computador.

III. MARCO TEÓRICO:

El lenguaje C se caracteriza por su capacidad de manipular directamente la memoria mediante punteros, lo que permite un control preciso sobre los recursos del sistema. Esta característica lo diferencia de lenguajes de más alto nivel y lo convierte en una herramienta fundamental para el desarrollo de sistemas operativos, controladores y software embebido.

Además, C presenta una sintaxis clara y concisa que facilita la comprensión del flujo lógico del programa. Su estructura modular favorece la reutilización de código y el mantenimiento de aplicaciones de gran tamaño, aspectos clave en proyectos de ingeniería.

Una de las principales fortalezas de C es su cercanía al hardware, lo que permite un acceso directo a la memoria y a los recursos del sistema. Esta característica lo convierte en un lenguaje ampliamente utilizado en el desarrollo de sistemas operativos, firmware y aplicaciones de tiempo real. A través del uso de punteros y estructuras de datos, C proporciona un alto nivel de control que no se encuentra comúnmente en lenguajes de más alto nivel.

Asimismo, el lenguaje C sirve como base conceptual para otros lenguajes de programación modernos. Lenguajes como C++, Java y Python adoptan parte de su sintaxis y principios fundamentales, lo que hace que el aprendizaje de C facilite la comprensión de otros entornos de desarrollo. Esta relación refuerza la importancia de C

como lenguaje introductorio en la formación académica en programación.

Desde el punto de vista educativo, el uso de C permite al estudiante comprender conceptos clave como la gestión de memoria, los tipos de datos y el flujo de ejecución de un programa. A diferencia de lenguajes interpretados, donde muchos procesos son abstractos, C expone de manera explícita las operaciones internas del sistema, fortaleciendo el razonamiento lógico y la comprensión del funcionamiento del computador.

a. El Lenguaje C: Origen y Filosofía

El lenguaje C fue desarrollado por Dennis Ritchie en los laboratorios Bell entre 1969 y 1973 como evolución del lenguaje B, creado para re implementar el sistema operativo UNIX. Su diseño sigue la filosofía de "confiar en el programador", ofreciendo control de bajo nivel cercano al hardware mientras mantiene una sintaxis de alto nivel estructurada. A diferencia de lenguajes interpretados, C es compilado, lo que implica que el código fuente se traduce directamente a lenguaje máquina, generando ejecutables nativos y altamente eficientes. Este paradigma lo convierte en un lenguaje de programación de sistemas, ideal para desarrollar sistemas operativos, controladores de dispositivos, firmware y aplicaciones donde el rendimiento y el control preciso de recursos son críticos. [1]

b. Estructuras de Control en Programación

Las estructuras de control determinan el flujo de ejecución de un programa. En programación estructurada –paradigma fundamental de C–, se clasifican en tres categorías fundamentales: [2]

- **Estructuras secuenciales:** Ejecutan instrucciones en orden lineal, una tras otra, sin alteraciones en el flujo. Constituyen la base sobre la cual se construyen las demás estructuras.
- **Estructuras condicionales:** Permiten tomar decisiones mediante la evaluación de expresiones lógicas. En C, esto se implementa principalmente con las sentencias if, if-else y switch, dirigiendo la ejecución por diferentes caminos según condiciones dadas.
- **Estructuras repetitivas (bucles):** Facilitan la ejecución iterativa de un bloque de código mientras o hasta que se cumpla una condición específica. C proporciona tres estructuras principales: *while* (comprueba la condición al inicio), *do-while* (comprueba la condición al final, garantizando al menos una ejecución) y *for* (ideal para iteraciones con un contador conocido).

A estas se suma la definición de funciones, un mecanismo esencial para la modularización. Las funciones permiten encapsular código reutilizable, dividir problemas complejos en subproblemas más manejables y organizar la lógica del programa. En C, toda función (con la excepción de main) debe ser declarada con un tipo de retorno explícito y sus parámetros deben tener tipos de datos definidos.

IV. Particularidades del Lenguaje C que Impactan sus Estructuras:

Una de las características más importantes del lenguaje C es el manejo explícito de memoria mediante el uso de punteros. A diferencia de lenguajes de alto nivel como Python, donde la gestión de memoria es automática, en C el programador tiene control directo sobre la asignación, acceso y liberación de memoria. Esto permite desarrollar aplicaciones más eficientes, pero también introduce riesgos como fugas de memoria y accesos inválidos si no se programa correctamente.

Los punteros permiten almacenar direcciones de memoria y acceder directamente a los datos almacenados en ellas. Esta característica resulta fundamental en el desarrollo de sistemas operativos, controladores de dispositivos y software embebido, donde el control del hardware es esencial. El uso adecuado de punteros contribuye a una mayor eficiencia en el uso de recursos, aunque requiere un mayor nivel de responsabilidad por parte del programador.

C presenta características de diseño fundamentales que condicionan y definen el uso de sus estructuras de control:

a. Tipado Estático y Explícito:

Cada variable, constante y parámetro de función debe declararse con un tipo de dato específico (*int, float, char,* punteros, etc.) antes de su uso. El compilador verifica la consistencia de tipos, lo que previene ciertos errores en tiempo de compilación y permite una asignación eficiente de memoria. Esto impone una disciplina de codificación más rigurosa y verbosa comparada con lenguajes de tipado dinámico.

b. Gestión Manual de Memoria:

C no dispone de un recolector de basura (*garbage collector*). La asignación y liberación de memoria, tanto estática (en la pila o *stack*) como dinámica (en el montón o *heap*), es responsabilidad explícita del programador mediante funciones como malloc, calloc y free. Este control absoluto es clave para la eficiencia y el desarrollo de sistemas, pero introduce riesgos como fugas de memoria (*memory leaks*) o accesos a memoria inválida.

c. El Concepto de Punteros:

Los punteros son variables cuyo valor es una dirección de memoria. Son un elemento central y distintivo de C. Permiten:

- Manipulación directa y eficiente de memoria.
- Implementación del paso por referencia en funciones, modificando variables originales.
- Trabajo con arreglos y cadenas de caracteres.
- Creación de estructuras de datos dinámicas complejas (listas, árboles, grafos). Su uso potencia al lenguaje, pero también es una fuente común de errores complejos, como desbordamientos de búfer o punteros colgantes (*dangling pointers*).

d. Paradigma de Programación Procedimental:

C es un lenguaje fundamentalmente procedimental. Organiza el código alrededor de funciones (procedimientos) que operan sobre datos, a diferencia del paradigma orientado a objetos donde los datos y el código se encapsulan juntos en objetos. El modularidad se logra mediante una cuidadosa arquitectura de funciones y el uso de archivos de cabecera (.h) y de implementación (.c).

e. Operadores de Bajo Nivel:

C incluye operadores que permiten un control a nivel de bits (como desplazamientos <<, >> y operaciones lógicas bit a bit &, |, ^), y operadores de acceso a memoria (* para desreferenciar, & para obtener dirección). Estas

capacidades, accesibles incluso dentro de las estructuras de control condicionales o repetitivas, subrayan la naturaleza de "ensamblador de alto nivel" de C. [3]

V. Desarrollo Técnico

Durante el desarrollo del proyecto se implementaron estructuras condicionales como if, else y switch, las cuales permiten tomar decisiones dentro del programa. Estas estructuras son fundamentales para controlar el flujo de ejecución y responder a diferentes condiciones de entrada.

Asimismo, se emplearon estructuras repetitivas como for, while y do-while, las cuales permiten ejecutar bloques de código de forma iterativa. El uso correcto de estos bucles contribuye a reducir la redundancia del código y mejorar su eficiencia. La correcta combinación de estructuras condicionales y repetitivas permitió resolver problemas de forma lógica y estructurada.

El desarrollo técnico presenta la implementación de las estructuras de control en el lenguaje C, analizando su sintaxis, comportamiento y aplicaciones prácticas. Se examinarán en orden creciente de complejidad: estructuras secuenciales, condicionales, repetitivas y de funciones, culminando con un ejercicio integrador.

A. Estructuras Secuenciales

Las estructuras secuenciales constituyen la forma más básica de control de flujo en programación. Consisten en la ejecución lineal de instrucciones, una después de la otra, en el orden exacto en que aparecen en el código fuente. No existen saltos, decisiones ni repeticiones. En C, esta secuencialidad es inherente al lenguaje: cada sentencia termina con punto y coma (;) y se ejecuta tras la anterior. Sirven para realizar cálculos directos, asignar valores, llamar funciones y preparar datos para estructuras más complejas.

5.1.1 Sintaxis en C:

La sintaxis básica no requiere palabras clave especiales más allá de las propias sentencias del lenguaje:

Fig. 1 Sintaxis en lenguaje C

```
tipo_de_dato variable = valor;      // Declaración y asignación
variable = expresion;               // Reasignación
funcion(argumentos);                // Llamada a función
```

Características clave en C:

- Declaración explícita de variables antes de su uso
- Asignación mediante el operador =
  - Las expresiones aritméticas y lógicas siguen precedencia de operadores
- Punto y coma obligatorio como terminador de sentencia

5.1.2 Ejemplo Práctico: Cálculo de Área y Perímetro

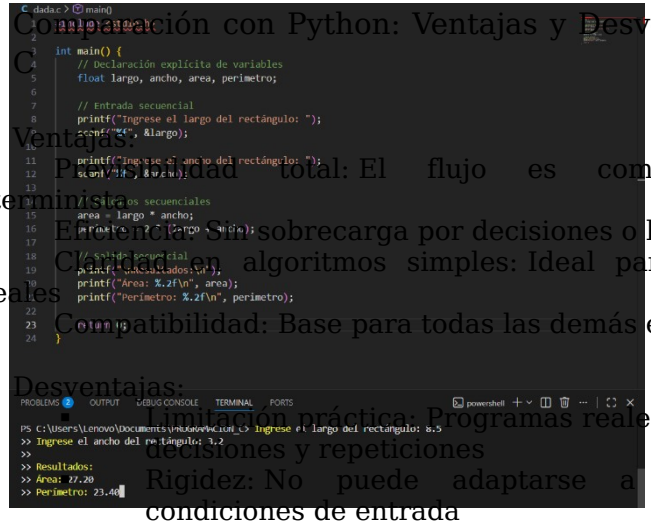


Fig. 2 Ejemplo de código en C para hallar el área de un circulo

5.1.3 Comparación con Python

Característica	***C***	***Python***
Declaración de varia	***Explícita con tipo***	***Implícita, dinámica***

5.1.4 Comparación con Python: Ventajas y Desventajas en C

Ventajas:

- Predictibilidad total: El flujo es completamente determinista
- Eficiencia: Sin sobrecarga por decisiones o bucles
- Claridad en algoritmos simples: Ideal para cálculos lineales
- Compatibilidad: Base para todas las demás estructuras

Desventajas:

- Limitación práctica: Programas reales requieren decisiones y repeticiones
  - Rigidez: No puede adaptarse a diferentes condiciones de entrada
  - Redundancia: Para tareas repetitivas, se necesita copiar código

5.1.5 Análisis de Impacto

- En C, la naturaleza compilada y estáticamente tipada añade una capa de rigurosidad a las estructuras secuenciales:

- El compilador verifica tipos en cada asignación
- Las conversiones implícitas pueden causar pérdida de precisión
- El orden de evaluación de expresiones está definido, evitando ambigüedades
- El manejo de memoria es explícito, incluso en flujos secuenciales simples

a. Estructuras Condicionales

¿Qué son y para qué sirven?

Las estructuras secuenciales constituyen la forma más básica de control de flujo en programación. Consisten en la ejecución lineal de instrucciones, una después de la otra, en el orden exacto en que aparecen en el código fuente. No existen saltos, decisiones ni repeticiones. En C, esta secuencialidad es inherente al lenguaje: cada sentencia termina con punto y coma (;) y se ejecuta tras la anterior. Sirven para realizar cálculos directos, asignar valores, llamar funciones y preparar datos para estructuras más complejas. [4]

i. Sintaxis en C:

C ofrece dos estructuras condicionales principales:
a) Sentencia if (simple y anidada):

```c
if (condicion) {
    // Bloque si condición es verdadera
}

if (condicion) {
    // Bloque si verdadera
} else {
    // Bloque si falsa
}

if (cond1) {
    // ...
} else if (cond2) {
    // ...
} else {
    // ...
}
```

Fig. 3 Sentencia if de la estructura condicional

b) **Sentencia** switch**:**

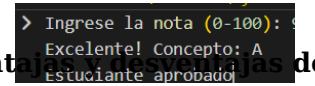
Fig. 4 Sentencia switch de la estructura condicional

ii. Ejemplo Práctico: Sistema de Calificaciones

```c
#include <stdio.h>

int main() {
    float nota;
    char concepto;

    printf("Ingrese la nota (0-100): ");
    scanf("%f", &nota);

    // Validación con if
    if (nota < 0 || nota > 100) {
        printf("Error: Nota fuera de rango\n");
        return 1;
    }

    // Asignación de concepto con if-else anidado
    if (nota >= 90) {
        concepto = 'A';
    } else if (nota >= 80) {
        concepto = 'B';
    } else if (nota >= 70) {
        concepto = 'C';
    } else if (nota >= 60) {
        concepto = 'D';
    } else {
        concepto = 'F';
    }

    // Mostrar resultado con switch
    switch (concepto) {
        case 'A':
            printf("Excelente. Concepto: A\n");
            break;
        case 'B':
            printf("Muy bien. Concepto: B\n");
            break;
        case 'C':
            printf("Aprobado. Concepto: C\n");
            break;
        case 'D':
            printf("Aprobado mínimo. Concepto: D\n");
            break;
        case 'F':
            printf("Reprobado. Debe repetir.\n");
            break;
        default:
            printf("Error en cálculo\n");
    }

    // Operador ternario para mensaje adicional
    (nota >= 60) ? printf("Estudiante aprobado\n") : printf("Estudiante reprobado\n");

    return 0;
}
```

Fig. 4 Ejemplo de la estructura condicional

iii. Ejemplo Práctico: Sistema de Calificaciones



iv. Ventajas y desventajas de C

Ventajas:

- Eficiencia de ejecución: Las condiciones se compilan a saltos directos en ensamblador
- Control preciso: switch optimizado para múltiples valores discretos
- Operadores bit a bit: Pueden usarse en condiciones para optimización
- Evaluación de cortocircuito: && y || evitan evaluar lo innecesario

Desventajas:

- Riesgo de "else colgante": Sin {}, solo la primera sentencia va en el if

```
if (condición)
    printf("Verdadero\n"); // Solo esto está en el if
printf("Siempre se ejecuta\n");  // ¡Fuera del if!
```

- Switch limitado: Solo funciona con tipos enteros (no cadenas ni objetos)
- Sin elif: Requiere anidación explícita con else if
- Peligro con = vs ==: if (x = 5) asigna, no compara (error común)

Característica	*C*	*Python*
Estructura básica	*if (cond) { }*	*if cond: (con indentación)*
Operadores lógicos	*&&, ||, !*	*and, or, not*
Switch/ Case	*switch con case y break*	*match (Python 3.10+) o if-elif*
Operador ternario	*cond ? expr1 : expr2*	*expr1 if cond else expr2*
Alcance de variables	*Dentro del bloque donde se declaran*	*Basado en indentación*

b. Estructuras repetitivas

i. ¿Qué es y para qué sirve?

Las **estructuras repetitivas** son mecanismos de control que permiten **ejecutar un mismo bloque de instrucciones varias veces**, siempre que se cumpla una condición determinada.

En el lenguaje C, estas estructuras permiten automatizar tareas repetitivas y controlar el flujo del programa de manera eficiente. [5]

Sirven para:

- Repetir operaciones sin duplicar código
- Procesar grandes cantidades de datos
- Realizar conteos, sumatorias y cálculos
- Implementar menús y validaciones
- Recorrer arreglos y matrices

Las principales estructuras repetitivas en C son:

- For
- While
- do-while

ii. Sintaxis en el lenguaje C

- **Estructura for**

```
for (inicialización; condición; incremento) {
    // instrucciones
}
```

- **Estructura while**

```
while (condición) {
    // instrucciones
}
```

- **Estructura so-while**

```
do {
    // instrucciones
} while (condición);
```

iii. Ejemplo práctico y funcional

Haremos un código para mostrar del 1 al 5 usando while:

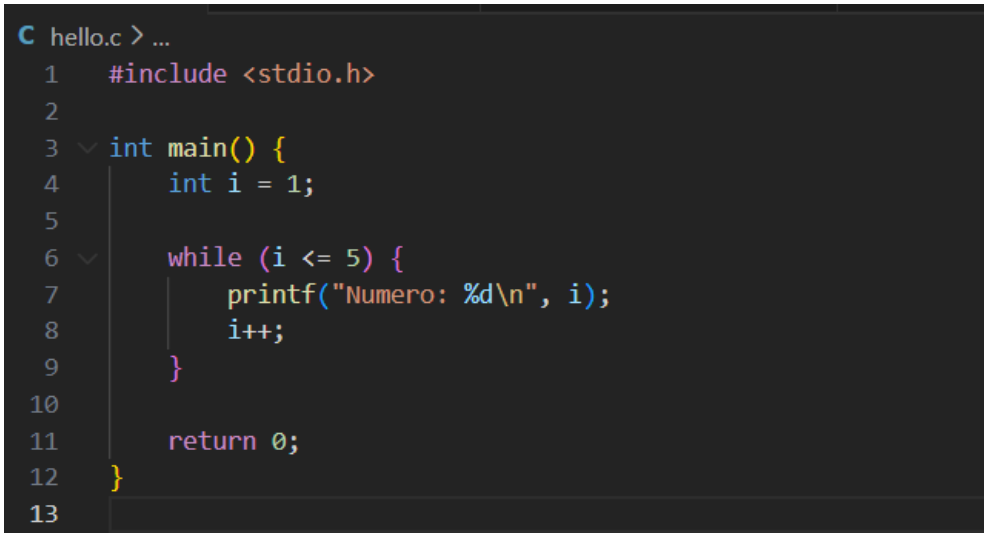


Fig. 5 Ejemplo de la estructura repetitiva

En el terminal tenemos o en la salida:

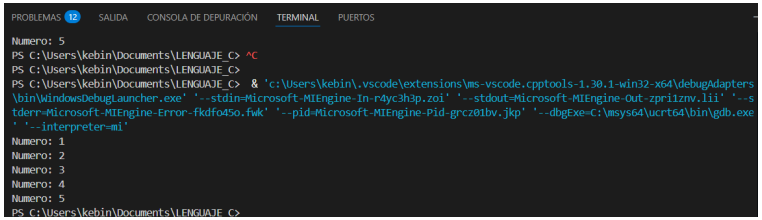


Fig. 6 Salida del ejemplo de la estructura repetitiva

iv. Comparación con Python (similitudes y diferencias)

Similitudes:

- **Ambos usan bucles para repetir instrucciones**
  En C y Python se utilizan bucles para contar, sumar, recorrer datos, etc.
- **Ambos tienen la estructura for y while**
  for → cuando hay un número definido de repeticiones
  while → cuando la repetición depende de una condición
- **La condición se evalúa de forma lógica**
  Si la condición es verdadera → el ciclo continúaSi es falsa → el ciclo termina

Diferencias:

- **Forma de escribir el código (sintaxis)**
  - **En C la sintaxis es más estricta:**
    Usa llaves {}
    Usa punto y coma;
    Se debe declarar el tipo de variable
  - **En Python la sintaxis es más simple:**
    No usa llaves
    No usa punto y coma
    No es necesario declarar el tipo de variable

v. Análisis de ventajas y desventajas del lenguaje C

Ventajas

- Alta eficiencia y velocidad de ejecución
- Control detallado del flujo del programa
- Uso eficiente de memoria
- Ideal para sistemas embebidos y aplicaciones de ingeniería

Desventajas:

- Sintaxis más compleja que Python
- Mayor posibilidad de errores
- No posee manejo automático de memoria
- Menor legibilidad para principiantes

VI. Definición

El análisis comparativo entre C y Python permite identificar diferencias significativas entre un lenguaje de bajo nivel y uno de alto nivel. El lenguaje C destaca por su eficiencia y control directo del hardware, lo que lo hace ideal para aplicaciones donde el rendimiento es crítico. Por otro lado, Python ofrece una sintaxis más simple y una mayor rapidez de desarrollo, siendo adecuado para aplicaciones de alto nivel y prototipado rápido.

En términos de manejo de memoria, C exige un control manual, lo que aumenta la complejidad pero mejora la eficiencia. Python, en cambio, gestiona la memoria automáticamente, reduciendo errores comunes pero sacrificando control. Esta comparación permite concluir que ambos lenguajes cumplen roles distintos y complementarios dentro del desarrollo de software.

a. ¿Qué es y para qué sirve?

Una función es un bloque de código que realiza una tarea específica y que puede ser reutilizado varias veces dentro de un programa.
En lugar de escribir el mismo código repetidamente, se agrupa dentro de una función y se llama cuando sea necesario.
Las funciones sirven para:

- Organizar mejor el código
- Reutilizar instrucciones
- Reducir errores
- Facilitar la lectura y mantenimiento del programa
- Dividir un programa grande en partes más pequeñas

b. Sintaxis en el lenguaje C

Estructura general de una función en C:

Fig. 7 estructura de la función

Partes de una función:

- **Tipo de dato**: indica el tipo de valor que retorna (int, float, void, etc.)
- **Nombre función**: nombre que identifica a la función
- **Parametros**: datos que recibe la función (opcional)
- **Return:** devuelve un valor (no se usa si la función es void)

Ejemplo práctico y funcional

Función que suma a dos números:

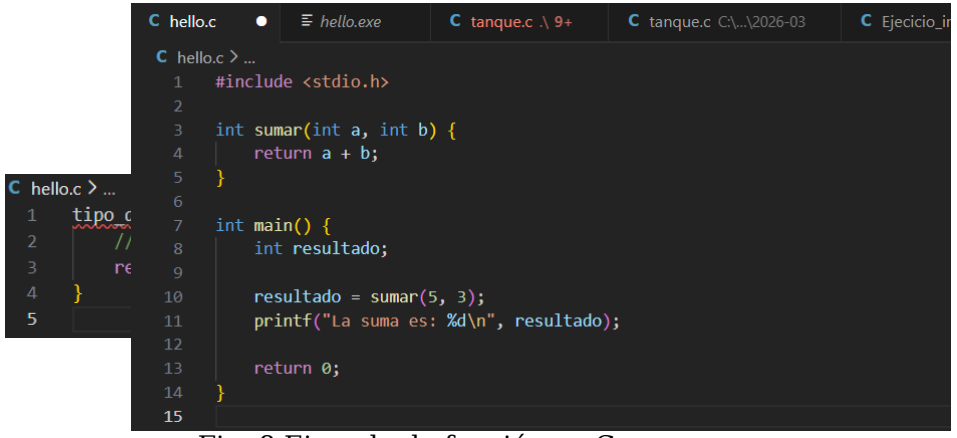


Fig. 8 Ejemplo de función en C

Salida:

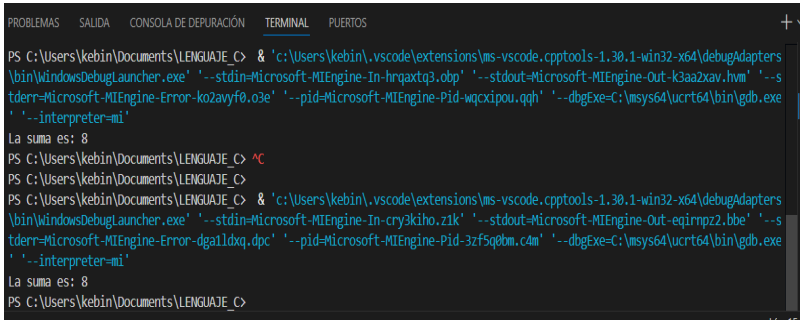

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS
PS C:\Users\kebin\Documents\LENGUAJE_C> & 'c:\Users\kebin\.vscode\extensions\ms-vscode.cpptools-1.30.1-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-hrqaxtq3.obp' '--stdout=Microsoft-MIEngine-Out-k3aa2xav.hvm' '--stderr=Microsoft-MIEngine-Error-ko2avyf0.o3e' '--pid=Microsoft-MIEngine-Pid-wqcxipou.qqh' '--dbgExe=C:\msys64\ucrt64\bin\gdb.exe' '--interpreter=mi'
La suma es: 8
PS C:\Users\kebin\Documents\LENGUAJE_C> ^C
PS C:\Users\kebin\Documents\LENGUAJE_C>
PS C:\Users\kebin\Documents\LENGUAJE_C> & 'c:\Users\kebin\.vscode\extensions\ms-vscode.cpptools-1.30.1-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-cry3kiho.z1k' '--stdout=Microsoft-MIEngine-Out-eqirnpz2.bbe' '--stderr=Microsoft-MIEngine-Error-dga1ldxq.dpc' '--pid=Microsoft-MIEngine-Pid-3zf5q0bm.c4m' '--dbgExe=C:\msys64\ucrt64\bin\gdb.exe' '--interpreter=mi'
La suma es: 8
PS C:\Users\kebin\Documents\LENGUAJE_C>

Fig. 9 Salida del ejemplo de función en C

c. Comparación con Python (similitudes y diferencias)

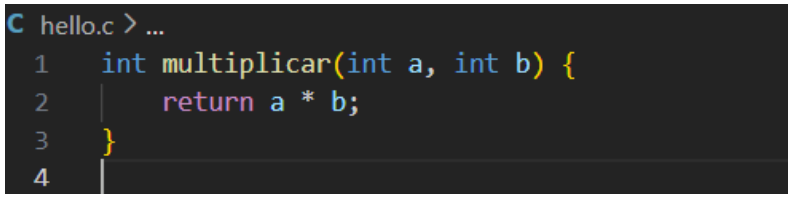
Similitudes:

- Ambos usan funciones para organizar y reutilizar código
- Ambas pueden recibir parámetros
- Ambas pueden devolver valores

Diferencias en C

- Se debe indicar el tipo de dato de retorno
- Los parámetros tienen tipo de dato
- Se usa return obligatoriamente si no es void

Fig. 10 Código en C



En Python:

.4.1.1.1. No se especifica tipo de retorno
.4.1.1.2. Sintaxis más simple

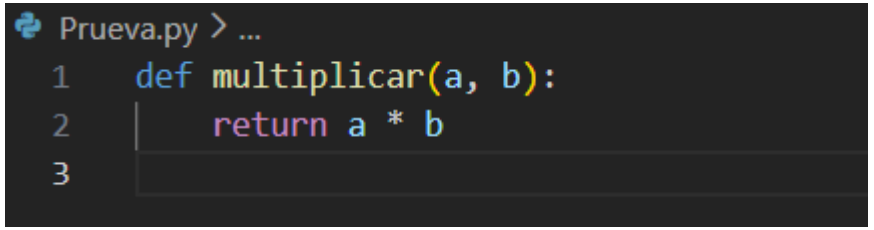


Fig. 11 Código en Python

d. Análisis de ventajas y desventajas del uso de funciones en C

- **Ventajas**
  - Código más organizado y modular
  - Reutilización de código
  - Mayor control del programa
  - Facilita el mantenimiento y depuración
- **Desventajas:**
  - Sintaxis más extensa que Python
  - Requiere definir tipos de datos
  - Puede resultar más complejo para principiantes

VII. Ejercicio integrador:

```c
#include <stdio.h>
float convertimospreciones(float psi)
{
    return psi * 6894.76;
}
float convertimosvolumenes(float m3)
{
    return m3 * 1000.0;
}
float convertimospotencias(float hp)
{
    return hp * 0.7457;
}
float convertimosareas(float ha)
{
    return ha * 10000;
}
int main()
{
    int menu;
    float entrada;
    float salida;
    printf("---BIENVENIDOS---\n");
    do
    {
        printf("---MENU DE OPCIONES---\n");
        printf("1.Converosr de preciones (PSI a
Pascales)\n");
        printf("2.Conversor de volumenes (m3 a Litros)\
n");
        printf("3.Conversor de potencia de (HP a kW)'\
n");
        printf("4.Conversor de cauldas (GPM a L/min)\
n");
        printf("5.Conversor de areas (ha a m2)\n");
        printf("6.Salir del menu\n");
        scanf("%d", &menu);
        switch (menu)
        {
        case 1:
            printf("Ingresa los psi:  ");
            scanf("%f", &entrada);
            if (entrada < 0)
            {
                printf("El valor ingresasdo es
incorrecto.\n");
            }
            else
            {
                salida = convertimospreciones(entrada);
                printf("la respuesta es: %.2f pascales\
n", salida);
            }
            break;
        case 2:
            printf("Ingresa el volumen en m3: ");
            scanf("%f", &entrada);
            if (entrada < 0)
            {
```

```
                printf("El valor ingresasdo es
incorrecto.\n");
            }
            else
            {
                salida = convertimosvolumenes(entrada);
                printf("la respuesta es: %.2f m3\n",
salida);
            }
            break;
        case 3:
            printf("ingrese su el valor en:");
            scanf("%f", &entrada);
            if (entrada < 0)
            {
                printf("El valor ingresasdo es
incorrecto.\n");
            }
            else
            {
                salida = convertimospotencias(entrada);
                printf("la respuesta es: %.2f hp\n",
salida);
            }
            break;
        case 4:
            printf("Ingrese el cauda en GPM: ");
            scanf("%f", &entrada);
            if (entrada < 0)
            {
```

```
                printf("El valor ingresasdo es
incorrecto.\n");
            }
            else
            {
                salida = convertimospreciones(entrada);
                printf("la respuesta es: %.2f L/min\n",
salida);
            }
            break;
        case 5:
            printf("Ingrese su area en ha:  ");
            scanf("%f", &entrada);
            if (entrada < 0)
            {
                printf("El valor ingresasdo es
incorrecto.\n");
            }
            else
            {
                salida = convertimosareas(entrada);
                printf("la respuesta es: %.2f m2\n",
salida);
            }
            break;
        case 6:
            printf("GRACIAS POR TU TIEMPO\n");
            break;
        }
```

```
    } while (menu != 6);
    return 0;
```

Fig. 11 Código en Python que integra las 3 estructuras vistas

VIII. Conclusiones y Reflexiones

En el presente trabajo se analizaron e implementaron las estructuras fundamentales de programación del lenguaje C, incluyendo las estructuras secuenciales, condicionales, repetitivas y el uso de funciones. A partir de su aplicación práctica, se evaluó su sintaxis, eficiencia y nivel de control, evidenciando que C ofrece un manejo detallado de los recursos del sistema, lo cual resulta ventajoso en aplicaciones donde el rendimiento es un factor crítico.

La explicación conceptual y la implementación de ejemplos funcionales permitieron comprender con mayor claridad el comportamiento de cada estructura de control en C, destacando la importancia del manejo explícito de memoria y de los tipos de datos. Este enfoque facilita una comprensión más profunda del funcionamiento interno de los programas y del computador, en comparación con lenguajes de más alto nivel.

Asimismo, se identificaron ventajas como la eficiencia y el control de bajo nivel, así como desventajas relacionadas con la mayor verbosidad del código y los riesgos asociados al manejo directo de memoria. En la comparación con Python, se observó que, si bien este último ofrece mayor simplicidad y rapidez de desarrollo, C proporciona una base más sólida para entender los fundamentos de la programación.

Finalmente, el desarrollo del proyecto permitió sintetizar los principales aprendizajes y dificultades del proceso, concluyendo que el lenguaje C es idóneo como lenguaje base para la formación en programación, ya que fortalece el razonamiento lógico, el diseño de algoritmos y la comprensión del funcionamiento interno del software y del hardware.

a. Dificultades encontradas:

Durante el desarrollo del presente trabajo práctico sobre las estructuras de control en el lenguaje C, se identificaron diversas dificultades propias del proceso de aprendizaje de un lenguaje de programación de bajo nivel. Una de las principales dificultades fue la rigurosidad sintáctica del lenguaje C, ya que requiere una correcta declaración de variables, uso preciso de símbolos como llaves, punto y coma, así como una adecuada gestión de tipos de datos. A diferencia de otros lenguajes más interpretativos, cualquier error mínimo en la sintaxis genera fallos en la compilación.

Asimismo, otra dificultad relevante fue la comprensión inicial del flujo de ejecución del programa, especialmente al combinar estructuras condicionales, repetitivas y funciones dentro de un mismo programa. La correcta organización del código y el control de la lógica del programa demandaron un análisis cuidadoso para evitar errores de ejecución y resultados incorrectos.

Finalmente, el uso de funciones implicó comprender el paso de parámetros, el alcance de las variables y el retorno de valores, aspectos que al inicio resultaron complejos, pero que con la práctica permitieron estructurar programas más claros y ordenados.

b. Ventajas y desventajas del lenguaje seleccionado:

- **Ventajas**

El lenguaje C presenta importantes ventajas desde el punto de vista académico y técnico. Entre las principales ventajas se encuentra su alta eficiencia y rapidez de ejecución, lo que permite desarrollar

programas con un uso óptimo de los recursos del sistema. Además, C es un lenguaje estructurado que fomenta el pensamiento lógico y ordenado, aspecto fundamental para la formación en programación.

Otra ventaja significativa es que el lenguaje C constituye la base de muchos otros lenguajes modernos, lo cual facilita posteriormente el aprendizaje de lenguajes como C++, Java o incluso Python desde una perspectiva más profunda.

- **Desventajas**

No obstante, también presenta desventajas. Una de ellas es su menor tolerancia a errores, ya que no cuenta con mecanismos automáticos de manejo de excepciones ni control de errores en tiempo de ejecución tan avanzados como otros lenguajes. Asimismo, la gestión manual de ciertos aspectos del programa puede resultar compleja para estudiantes que recién se inician en la programación.

c. Aprendizajes significativos obtenidos:

El desarrollo de este trabajo permitió adquirir aprendizajes significativos relacionados con el control del flujo de un programa, comprendiendo cómo las estructuras secuenciales, condicionales y repetitivas determinan el comportamiento de una aplicación.

Se logró comprender la importancia de las estructuras condicionales para la toma de decisiones dentro de un programa, así como el uso adecuado de las estructuras repetitivas para automatizar procesos y evitar la repetición innecesaria de código. Asimismo, la definición y uso de funciones permitió modular el programa, mejorar su legibilidad y facilitar su mantenimiento.

Otro aprendizaje relevante fue la comparación con el lenguaje Python, lo que permitió identificar similitudes conceptuales y diferencias prácticas, reforzando la comprensión de que, aunque los lenguajes puedan variar en sintaxis, los principios de la programación estructurada se mantienen.

d. Opinión personal sobre el lenguaje:

Desde una perspectiva académica, el lenguaje C resulta altamente formativo, ya que obliga al estudiante a comprender en profundidad la lógica del programa y el funcionamiento interno del código. Aunque su aprendizaje inicial puede resultar exigente, esta dificultad contribuye al desarrollo de habilidades analíticas y de resolución de problemas.

En opinión personal, el lenguaje C es una herramienta fundamental para la formación en programación, especialmente en carreras de ingeniería, ya que proporciona una base sólida para el entendimiento de otros lenguajes y tecnologías. Su estudio fortalece la disciplina, la precisión y el razonamiento lógico, competencias que resultan esenciales en la formación profesional.

En conclusión, el trabajo realizado permitió valorar al lenguaje C como un lenguaje robusto, estructurado y formativo, cuyo aprendizaje aporta conocimientos fundamentales para el desarrollo de programas eficientes y bien estructurados.

IX. Referencias

[1] E. Vieito, 17 Julio 2025. [En línea]. Available: https://dinahosting.com/blog/historia-del-lenguaje-de-programacion-c-por-que-sigue-estando-tan-presente/.

[2] R. Maldonado, 12 2 2023. [En línea]. Available: https://keepcoding.io/blog/que-son-estructuras-de-control-en-programacion/.

[3] Wikipedia, 12 3 2024. [En línea]. Available: https://es.wikipedia.org/wiki/Anexo:Operadores_de_C_y_C%2B%2B.

[4] H. Delgado, 23 4 2016. [En línea]. Available: https://disenowebakus.net/condicionales.php.

[5] UNAM, 31 Agosto 2024. [En línea]. Available: https://repositorio-uapa.cuaed.unam.mx/repositorio/moodle/pluginfile.php/3086/mod_resource/content/1/UAPA-Estructuras-Repeticion/index.html.

X. Videos consultados:

- https://www.youtube.com/watch?v=b_f_5Qj9FrU&list=PLO8lO9oepSLvAvnxCn_uCsFVCAYCDsGTK&index=5
- https://www.youtube.com/watch?v=b_f_5Qj9FrU&list=PLO8lO9oepSLvAvnxCn_uCsFVCAYCDsGTK&index=5
- https://www.youtube.com/watch?v=1-qjMS6C960&list=LL&index=2&t=601s

XI. ARTÍCULOS Y REVISTAS WEB

- Revista del repositorio de la PUCP: Lenguaje de programación en C
  https://hdl.handle.net/20.500.14657/181826

- Revista del repositorio de la upct: Informática aplicada: programación en lenguaje C
  http://hdl.handle.net/10317/1361

XII. ANEXOS:

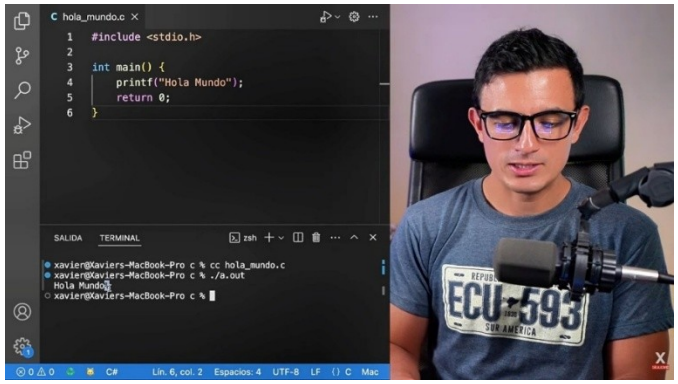
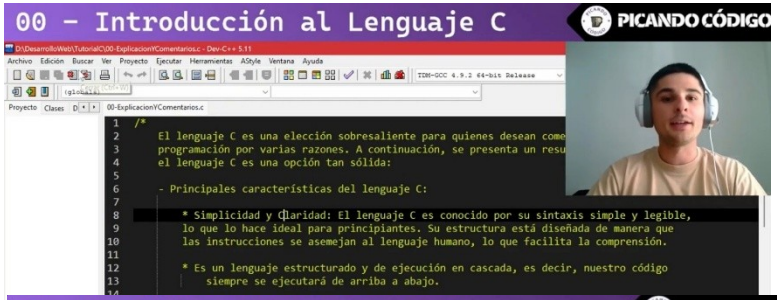


Fig. 12 Captura de pantalla de video consultado

```c
#define ABCD 2
#include <stdio.h>

int main(void)
{

#ifdef ABCD
    printf("1: yes\n");
#else
    printf("1: no\n");
#endif

#ifndef ABCD
    printf("2: no1\n");
#elif ABCD == 2
    printf("2: yes\n");
#else
    printf("2: no2\n");
#endif

#if !defined(DCBA) && (ABCD < 2 * 4 - 3)
    printf("3: yes\n");
#endif

// C23 directives #elifdef/#elifndef
#ifdef CPU
    printf("4: no1\n");
#elifdef GPU
    printf("4: no2\n");
#elifndef RAM
    printf("4: yes\n"); // selected in C23 mode, may be selected in pre-C23 mode
#else
    printf("4: no3\n"); // may be selected in pre-C23 mode
#endif
}
```

Fig. 13 Captura de pantalla de video consultado



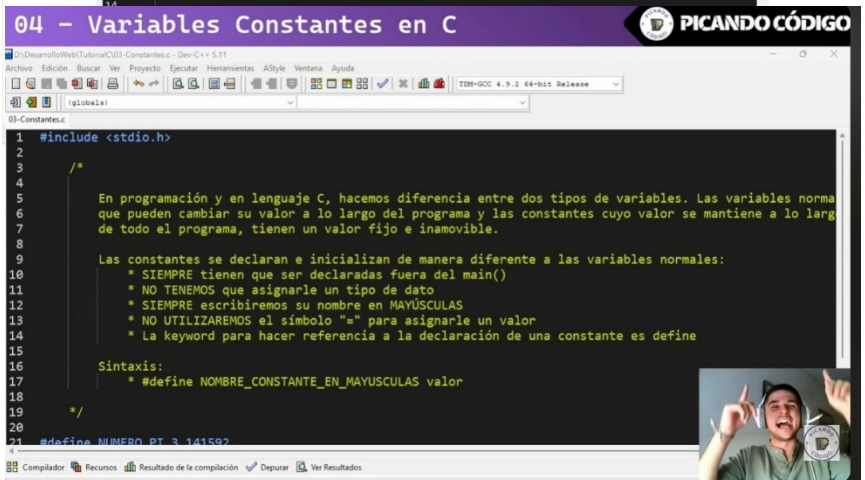


Fig. 14 Captura de pantalla de video consultado

Fig. 15 Captura de pantalla de video consultado

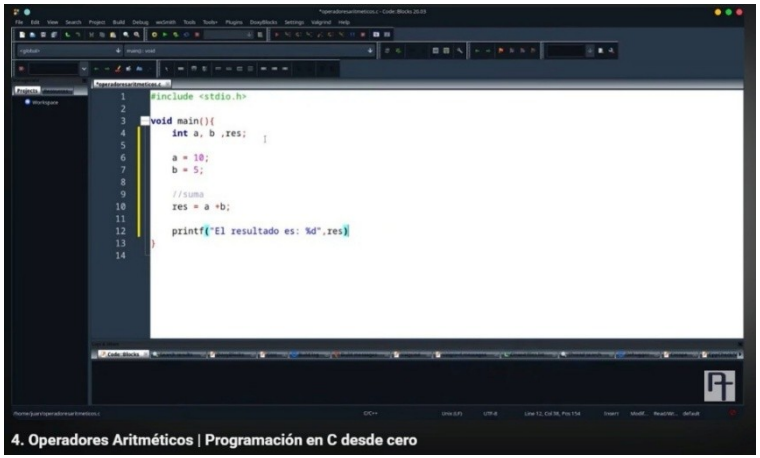
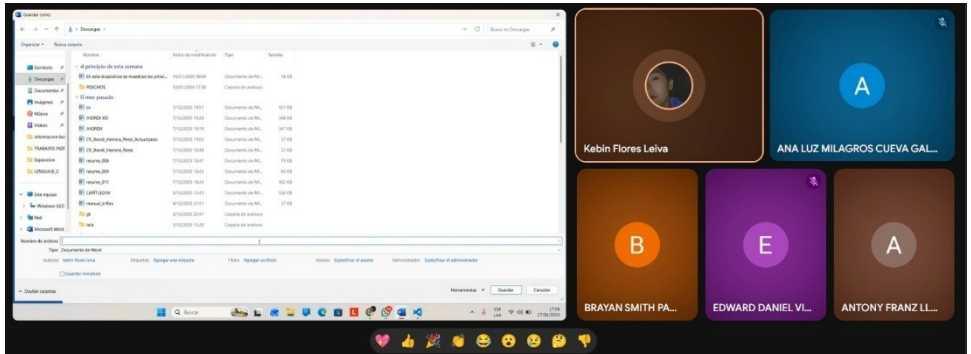


Fig. 16 Captura de pantalla de video consultado

Fig. 17 Captura de pantalla de video consultado

Fig. 18 Captura de pantalla de reunión virtual de los integrantes discutiendo el trabajo



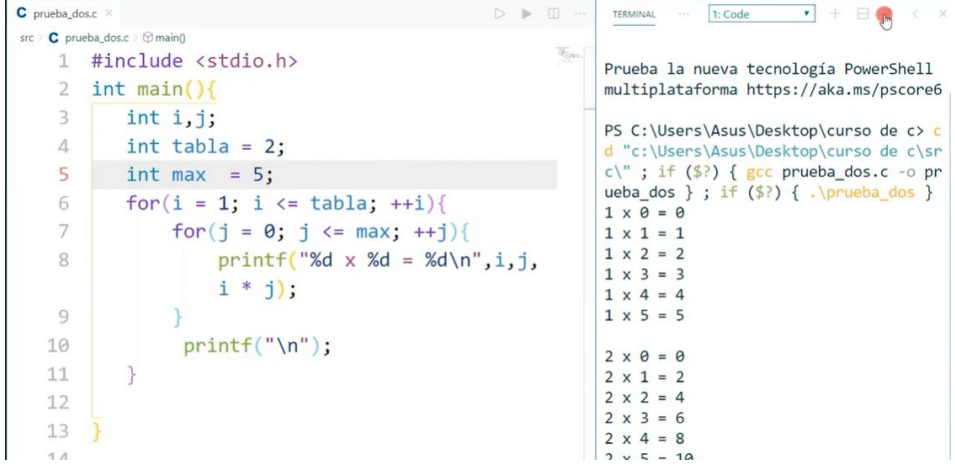


Fig. 19 Captura de pantalla de video consultado

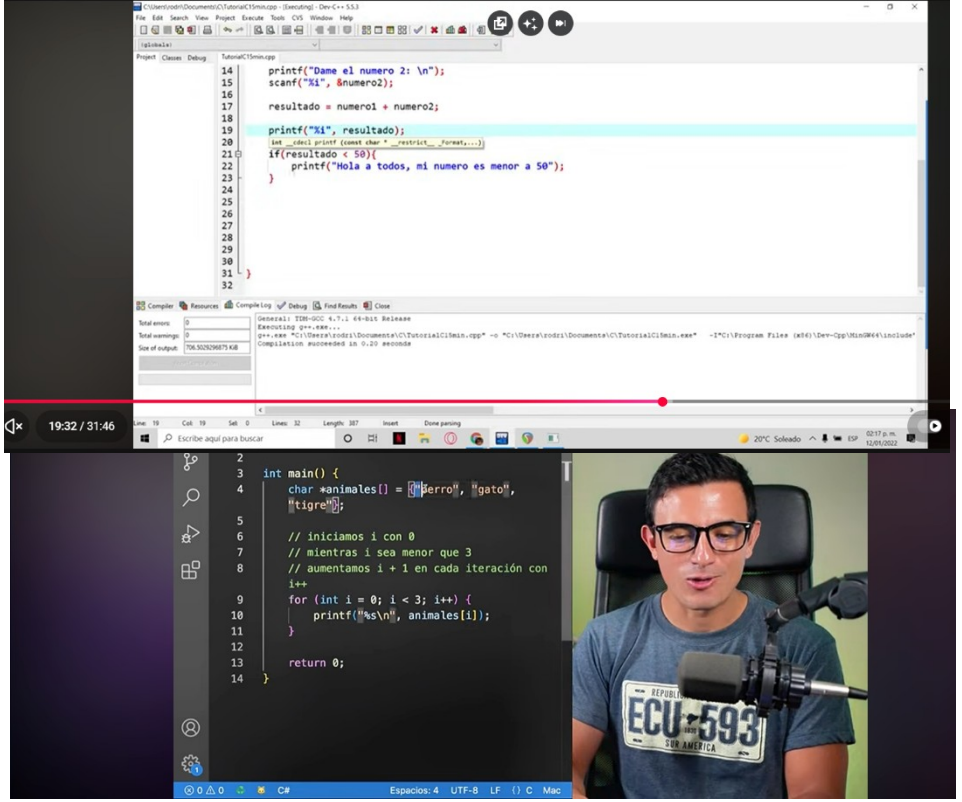


Fig. 20 Captura de pantalla de video del tema

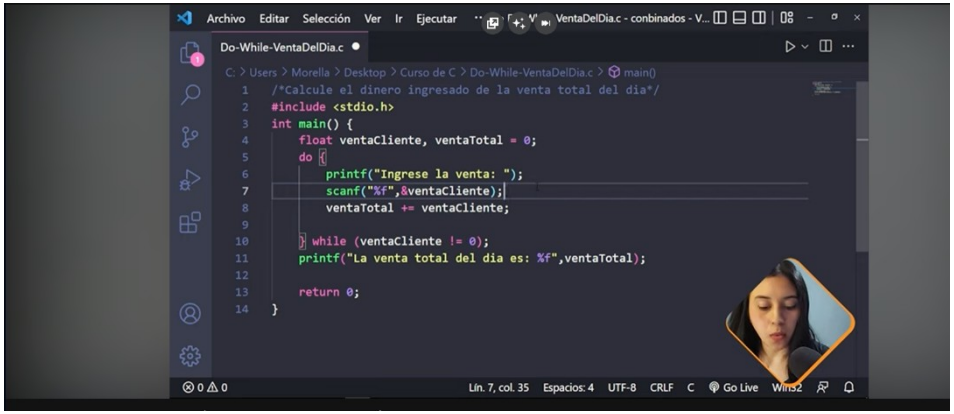


Fig. 17 captura de video consultado sobre el tema